**Chapter 12: Word2Vec, Doc2Vec, and Gensim**

Chapter 12 focuses on generating vector representations of words and documents using **Word2Vec** and **Doc2Vec**, popular models implemented in the **Gensim library**. These vector representations, known as embeddings, encode semantic meaning and are foundational for many natural language processing (NLP) tasks.

Word embeddings are dense, low-dimensional vector representations of words in a continuous vector space. They capture the semantic meaning of words based on their context in a corpus.

* **Key Characteristics**:
  + Words with similar meanings or usage are mapped to nearby points in the vector space.
  + Unlike sparse one-hot encodings, embeddings are dense and distributed, allowing for richer representations.

**Key Topics Covered**

1. **Word2Vec**:
   * **Training Methods**:
     + **Skip-gram**: Predicts surrounding words given a target word.
     + **CBOW (Continuous Bag of Words)**: Predicts a target word given surrounding words.
   * **Parameters**:
     + size: Dimensionality of vectors.
     + window: Context window size.
     + min\_count: Minimum frequency of words to be included in training.
     + sg: Specifies Skip-gram (1) or CBOW (0).

**Dimensions in Word2Vec:**

* The **dimensionality** of the word vectors refers to the size of the vector space where each word is represented.
* It is specified using the parameter vector\_size (or size in older versions) during the training of a Word2Vec model.

**Common Values:**

* Typical dimensions are **100, 200, or 300**, but they can be set to other values depending on the use case.
* A **higher dimension** can capture more semantic details but requires more computation and memory.
* A **lower dimension** might be computationally faster but could lose finer-grained semantic information.

**Feature Representation:**

Each word is represented as a dense vector in the embedding space. For example:

* In a 100-dimensional space, a word like "king" might be represented as:
* [0.2, -0.3, 0.8, ..., 0.5]
* The vectors capture semantic and syntactic relationships. For example:
* vector("king") - vector("man") + vector("woman") ≈ vector("queen")

**Trade-offs:**

* **Smaller dimensions**: Useful for faster computation but might not capture complex relationships.
* **Larger dimensions**: Capture more detailed relationships but risk overfitting and require more resources.

1. **Doc2Vec**:
   * Extends Word2Vec to generate embeddings for entire documents or paragraphs.
   * **Training Modes**:
     + **PV-DM (Distributed Memory)**: Uses context to predict words and learn document embeddings.
     + **PV-DBOW (Distributed Bag of Words)**: Learns document embeddings without considering word order.
   * Applications include text classification, clustering, and semantic similarity tasks.
2. **Gensim's Implementation**:
   * Tools for training and saving embeddings.
   * Pre-trained models for common NLP tasks.

**Practical Implementation**

**1. Using Word2Vec with Gensim**

**Import and Train:**

from gensim.models import Word2Vec

from gensim.utils import simple\_preprocess

# Sample Corpus

corpus = [

"This is the first document.",

"This document is the second document.",

"And this is the third one.",

"Is this the first document?"

]

# Preprocessing

sentences = [simple\_preprocess(doc) for doc in corpus]

# Train Word2Vec Model

model = Word2Vec(sentences, vector\_size=100, window=5, min\_count=1, sg=0)

# Access Word Vectors

print(model.wv['document']) # Vector for the word 'document'

**Evaluate Word Similarities:**

# Most similar words to 'document'

print(model.wv.most\_similar('document'))

**Save and Load Models:**

model.save("word2vec.model")

loaded\_model = Word2Vec.load("word2vec.model")

**2. Using Doc2Vec**

**Tags in Doc2Vec**

In **Doc2Vec**, tags are unique identifiers assigned to each document during training. These tags allow the model to associate a vector representation with a specific document. The tags are crucial because:

* They enable the model to learn document embeddings.
* Each document is represented as a unique vector based on its content and tag.

**Tag Formats**

1. **Numeric Tags**:
   * Commonly, tags are integers corresponding to the index of the document in the dataset.
   * Example:
     + Document 0 → Tag [0]
     + Document 1 → Tag [1]
2. **Custom Tags**:
   * Tags can also be strings or a combination of strings and integers.
   * Useful when documents have meaningful identifiers like titles, categories, or IDs.
   * Example:
     + Document with ID "doc\_123" → Tag ["doc\_123"]
3. **Multiple Tags**:
   * A document can have multiple tags if needed, e.g., a unique identifier and a category.
   * Example:
     + Document about Machine Learning → Tags ["doc\_456", "Machine\_Learning"]

**Use Case in gensim.models.Doc2Vec**

Tags are defined using the TaggedDocument class in Gensim. Example:

from gensim.models.doc2vec import TaggedDocument

# Sample documents

documents = [

"This is the first document.",

"Another document about machine learning.",

"This is about embeddings."

]

# Tagging documents

tagged\_docs = [TaggedDocument(words=doc.split(), tags=[str(i)]) for i, doc in enumerate(documents)]

# TaggedDocs Example:

# TaggedDocument(words=['This', 'is', 'the', 'first', 'document'], tags=['0'])

# TaggedDocument(words=['Another', 'document', 'about', 'machine', 'learning'], tags=['1'])

**Importance of Tags**

* **Document Identification**: Tags uniquely identify each document, allowing the model to learn embeddings specific to them.
* **Inference**: Tags enable the model to retrieve the learned vector for a specific document after training.
* **Flexibility**: Using meaningful tags (e.g., document IDs or categories) allows easier interpretation and analysis of document embeddings.

**Import and Train:**

from gensim.models.doc2vec import Doc2Vec, TaggedDocument

# Sample Corpus

documents = [

"This is a document about NLP.",

"Another document discussing deep learning.",

"This document is on embeddings.",

"We study word and document embeddings."

]

# Tagging Documents

tagged\_docs = [TaggedDocument(simple\_preprocess(doc), [i]) for i, doc in enumerate(documents)]

# Train Doc2Vec Model

doc\_model = Doc2Vec(tagged\_docs, vector\_size=100, window=5, min\_count=1, epochs=20)

# Access Document Vector

print(doc\_model.dv[0]) # Vector for the first document

**Infer Vector for New Data:**

new\_doc = simple\_preprocess("A new document on embeddings.")

new\_vec = doc\_model.infer\_vector(new\_doc)

print(new\_vec)

**Homework:**

1. Preprocess a real-world dataset (e.g., [20 Newsgroups Dataset](https://scikit-learn.org/0.19/datasets/twenty_newsgroups.html)).
2. Train a **Word2Vec** model:
   * Evaluate most similar words for a target word.
   * Identify the odd one out in a list of words using doesnt\_match.
3. Train a **Doc2Vec** model:
   * Compute document embeddings.
   * Perform clustering of the document embeddings using KMeans or a similar technique.
4. Compare the performance of embeddings (Word2Vec vs. Doc2Vec) for document similarity.